CT5106

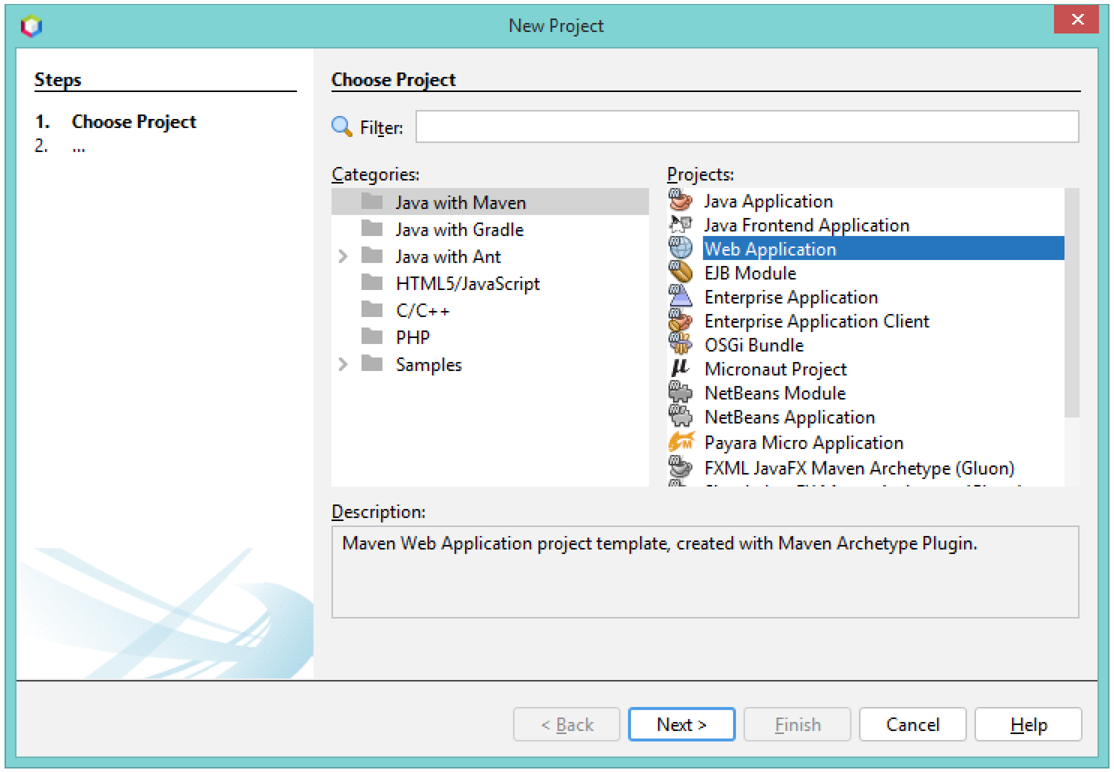
Java Server Faces (JSF)

JSF Overview

- JSF is an MVC framework for web applications
- Focused around connecting UI components / widgets with data sources and (server-side) event handlers
- JSF renders the UI components properly for the client type (in our case the web browser)

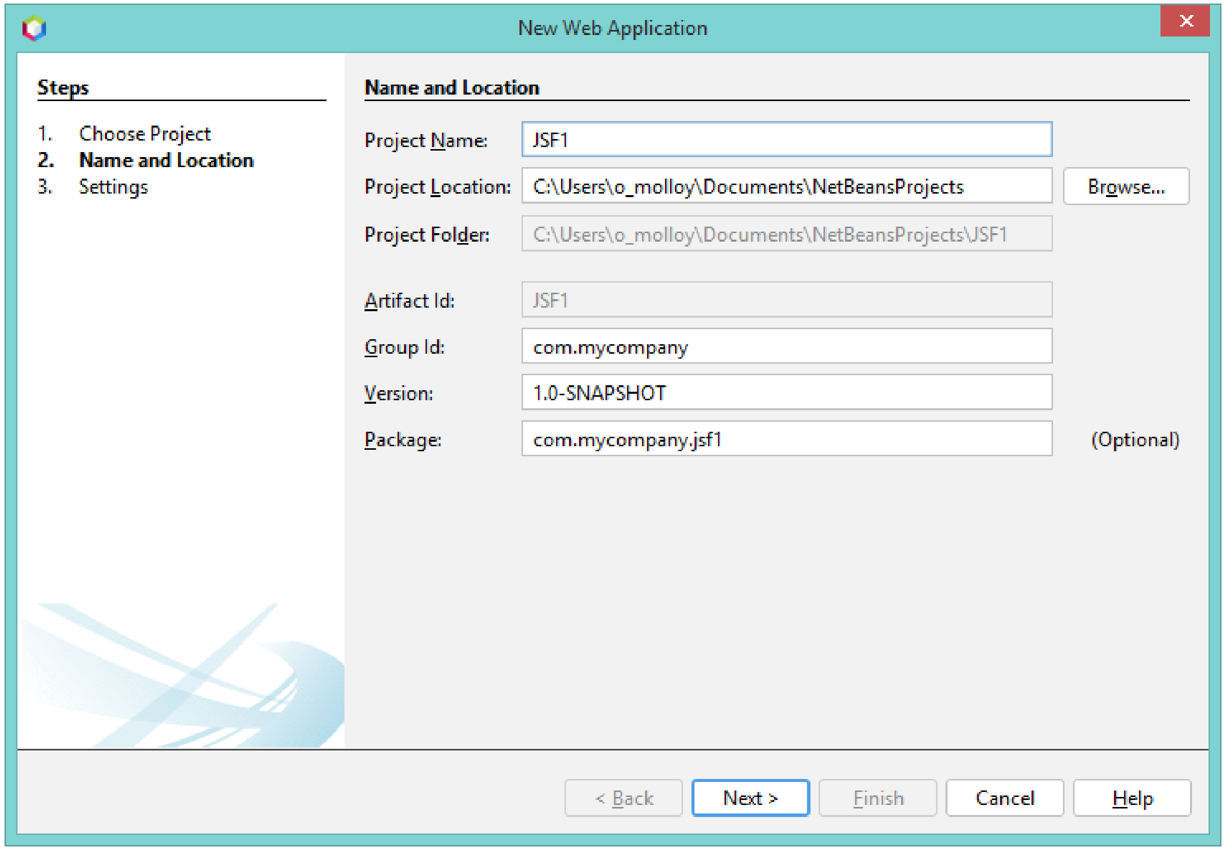
Setting up JSF on Apache NetBeans

- File / New Project / Java with Maven / Web Application



Maven Web Application project template, created with Maven Archetype Plugin.

- Enter project name, and accept other defaults



Name and Location

Project Name: JSF1

Project Location: C:\Users\o_molloy\Documents\NetBeansProjects

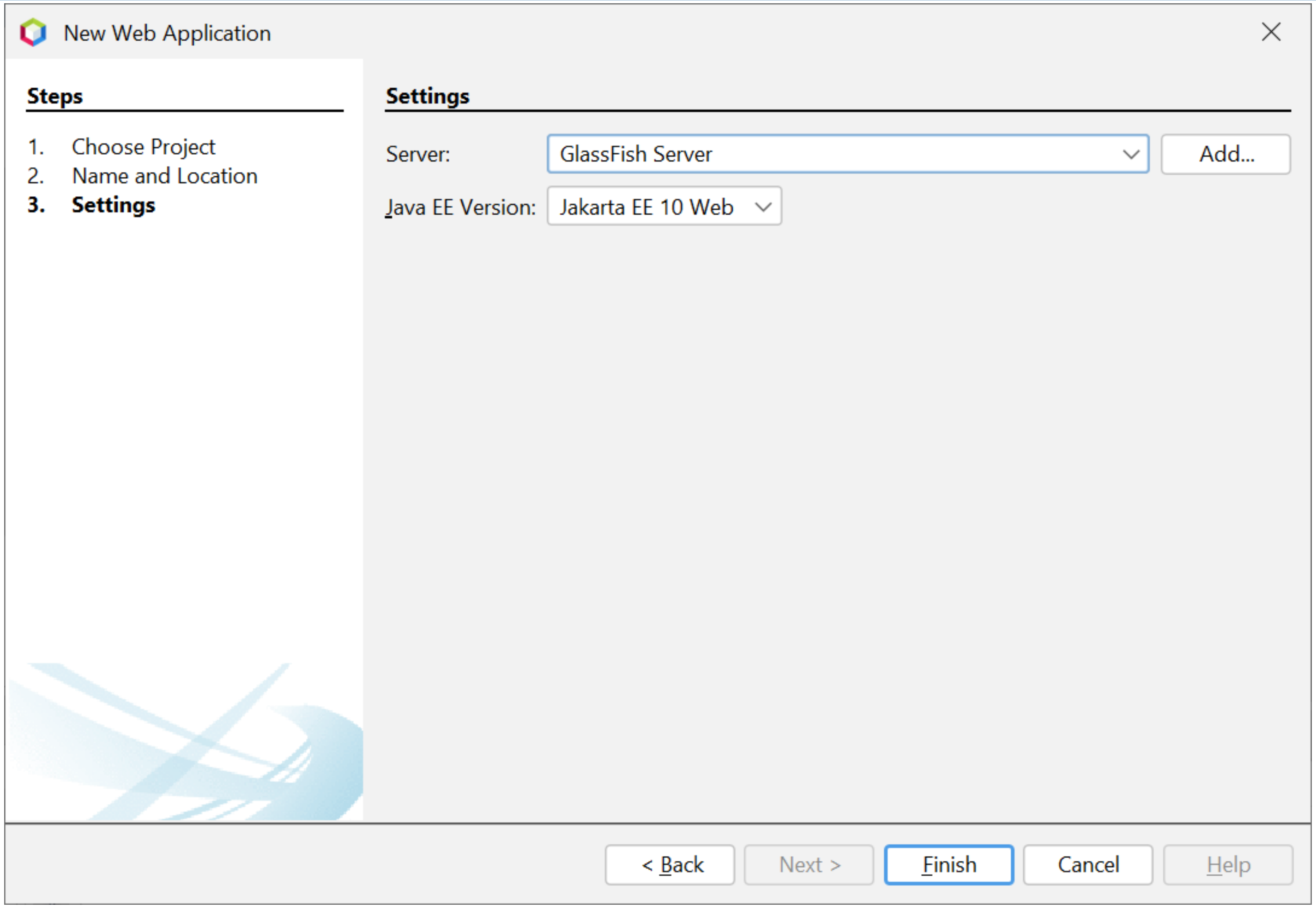
Project Folder: C:\Users\o_molloy\Documents\NetBeansProjects\JSF1

Artifact Id: JSF1

Group Id: com.mycompany

Version: 1.0-SNAPSHOT

Package: com.mycompany.jsf1 (Optional)



New Web Application

Steps

1. Choose Project
2. Name and Location
3. **Settings**

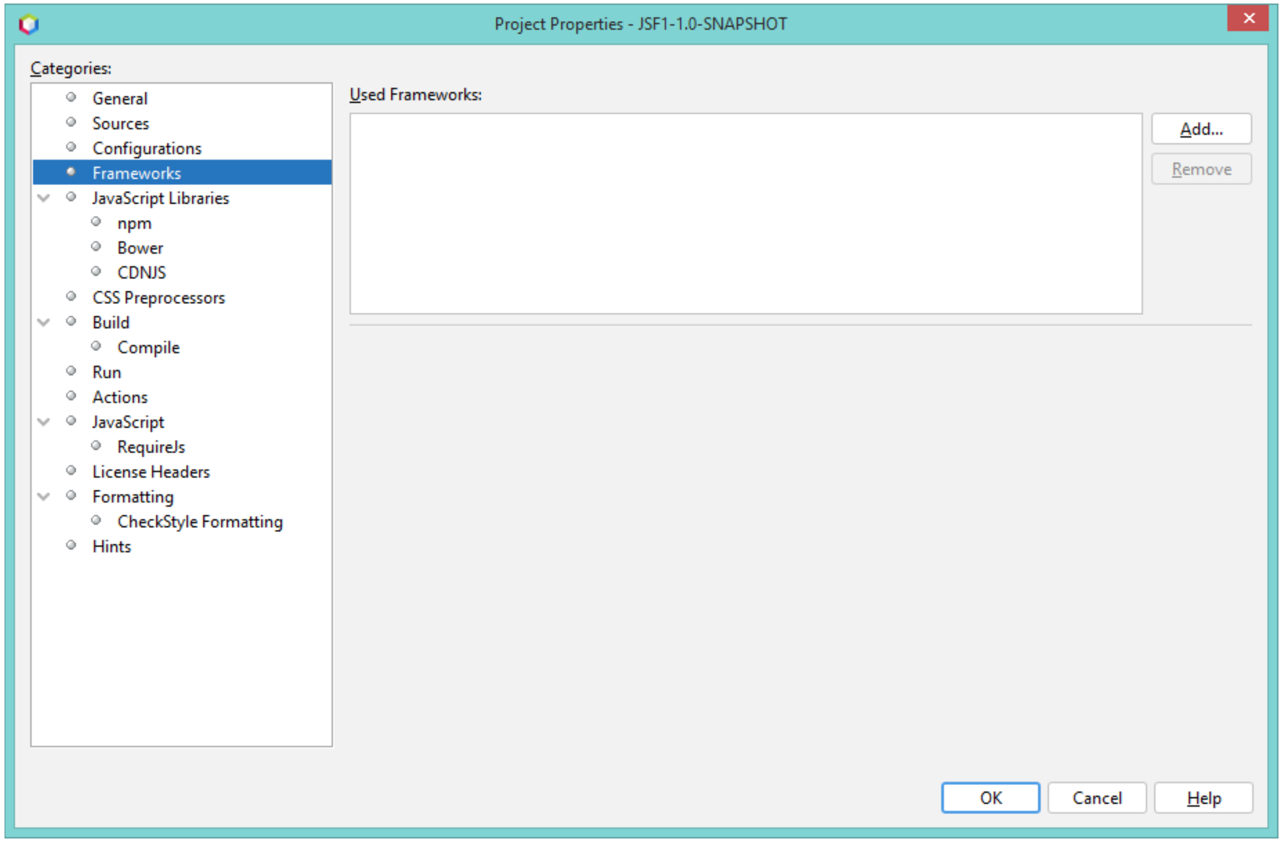
Settings

Server: GlassFish Server

Java EE Version: Jakarta EE 10 Web

Project Properties

- Right-click on project
- Select ‘Frameworks’ and click on ‘Add…’ button

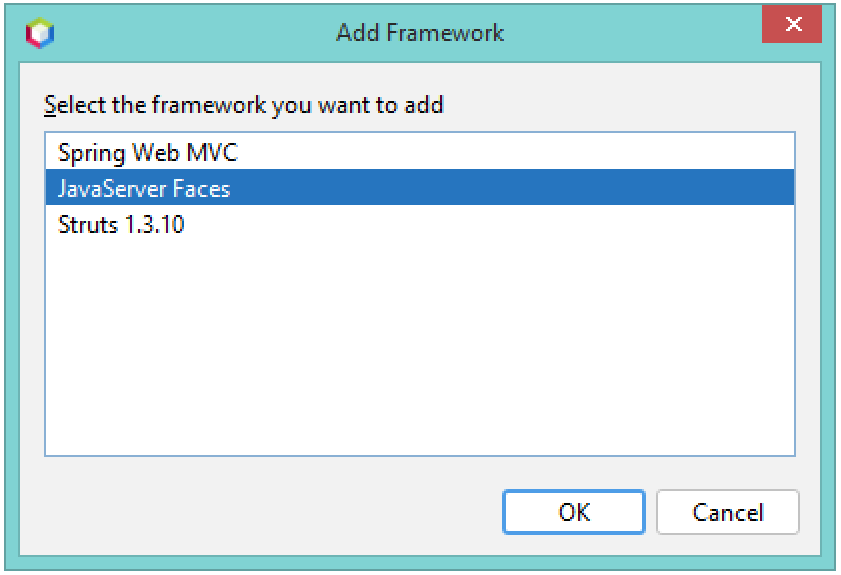

Project Properties - JSF1-1.0-SNAPSHOT

Categories:

- General
- Sources
- Configurations
- Frameworks
- JavaScript Libraries
  - npm
  - Bower
  - CDNJS
- CSS Preprocessors
- Build
  - Compile
- Run
- Actions
- JavaScript
  - RequireJs
- License Headers
- Formatting
  - CheckStyle Formatting
- Hints

Used Frameworks:

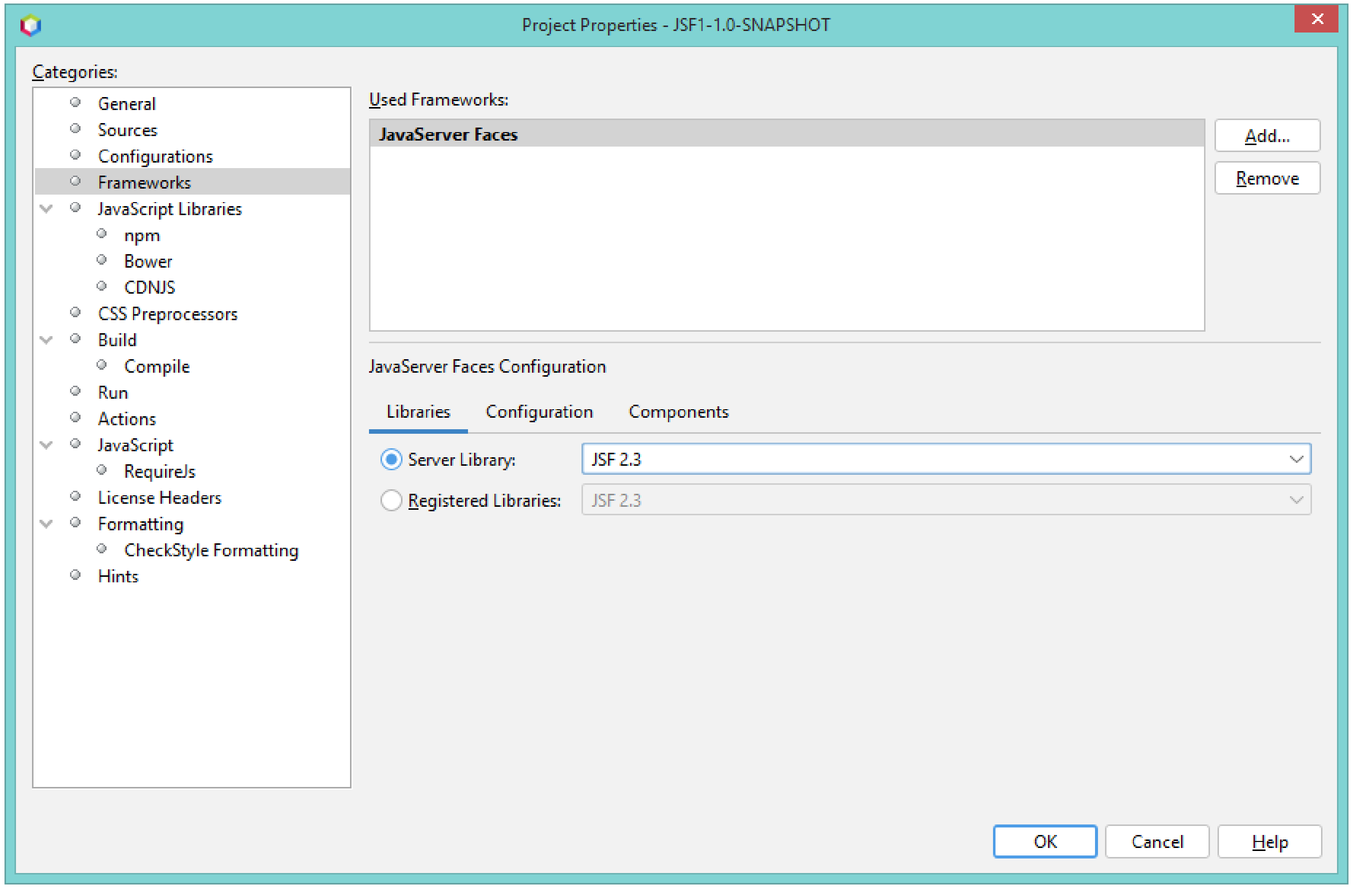
- Select JSF framework to add



Select the framework you want to add

Spring Web MVC
JavaServer Faces
Struts 1.3.10

- OK


Project Properties - JSF1-1.0-SNAPSHOT

Categories:
- General
- Sources
- Configurations
- Frameworks
- JavaScript Libraries
  - npm
  - Bower
  - CDNJS
- CSS Preprocessors
- Build
  - Compile
- Run
- Actions
- JavaScript
  - RequireJs
- License Headers
- Formatting
  - CheckStyle Formatting
- Hints

Used Frameworks:

JavaServer Faces

JavaServer Faces Configuration

Libraries | Configuration | Components

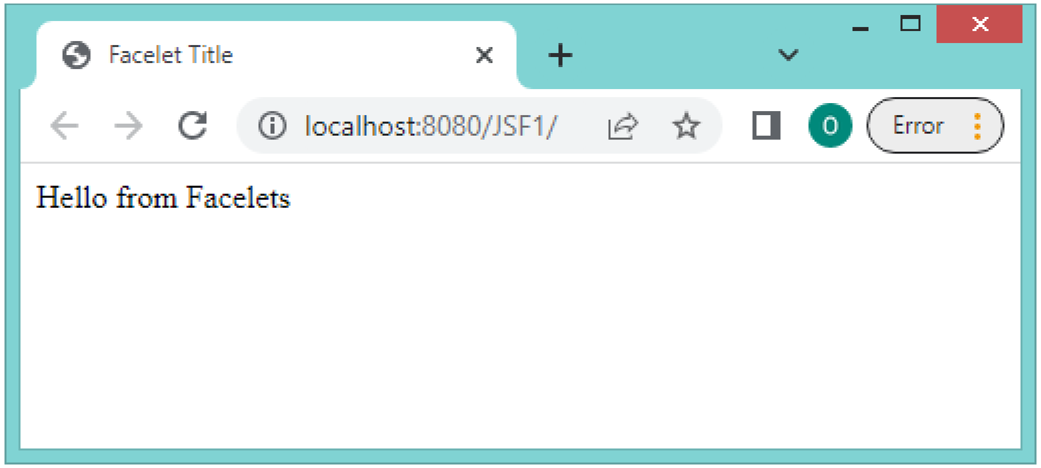
Server Library: JSF 2.3 [✓]

Registered Libraries: JSF 2.3

Then build

- Right-click on project and select ‘Build with Dependencies’
- If you see ‘BUILD SUCCESS’ then it is OK

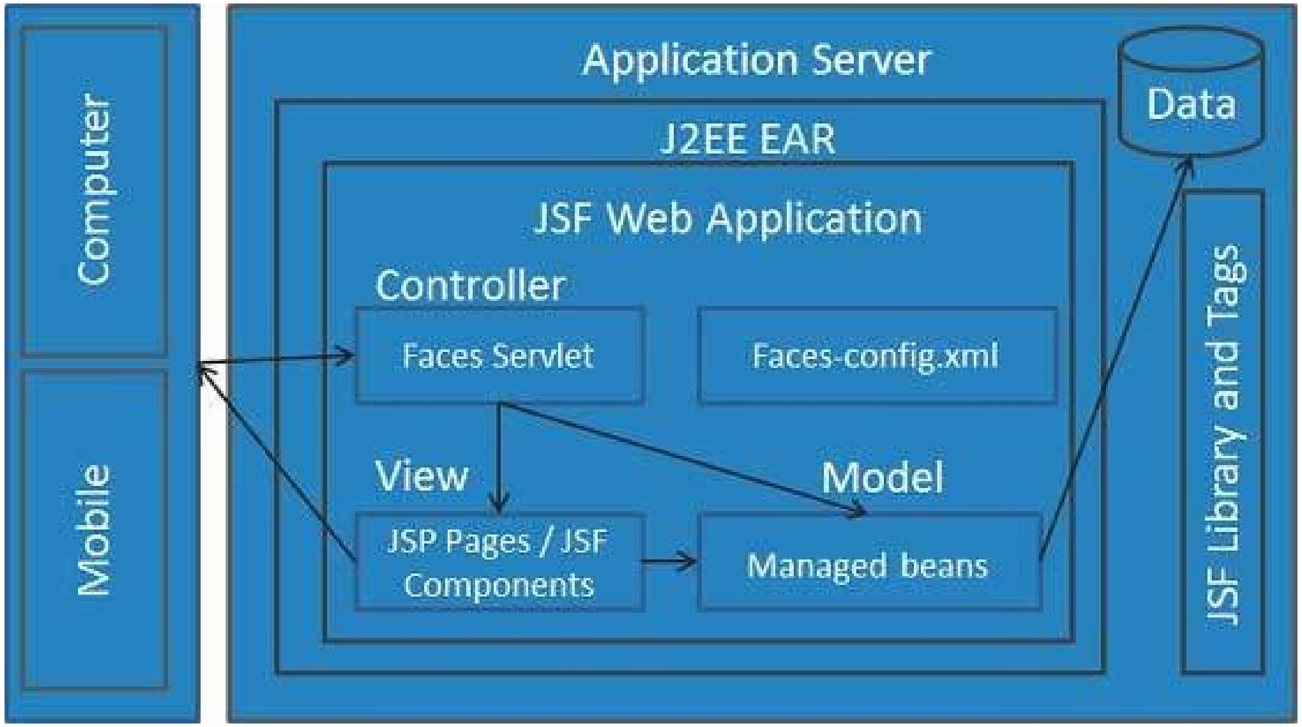
- Then run the project – should see this in browser:



Hello from Facelets

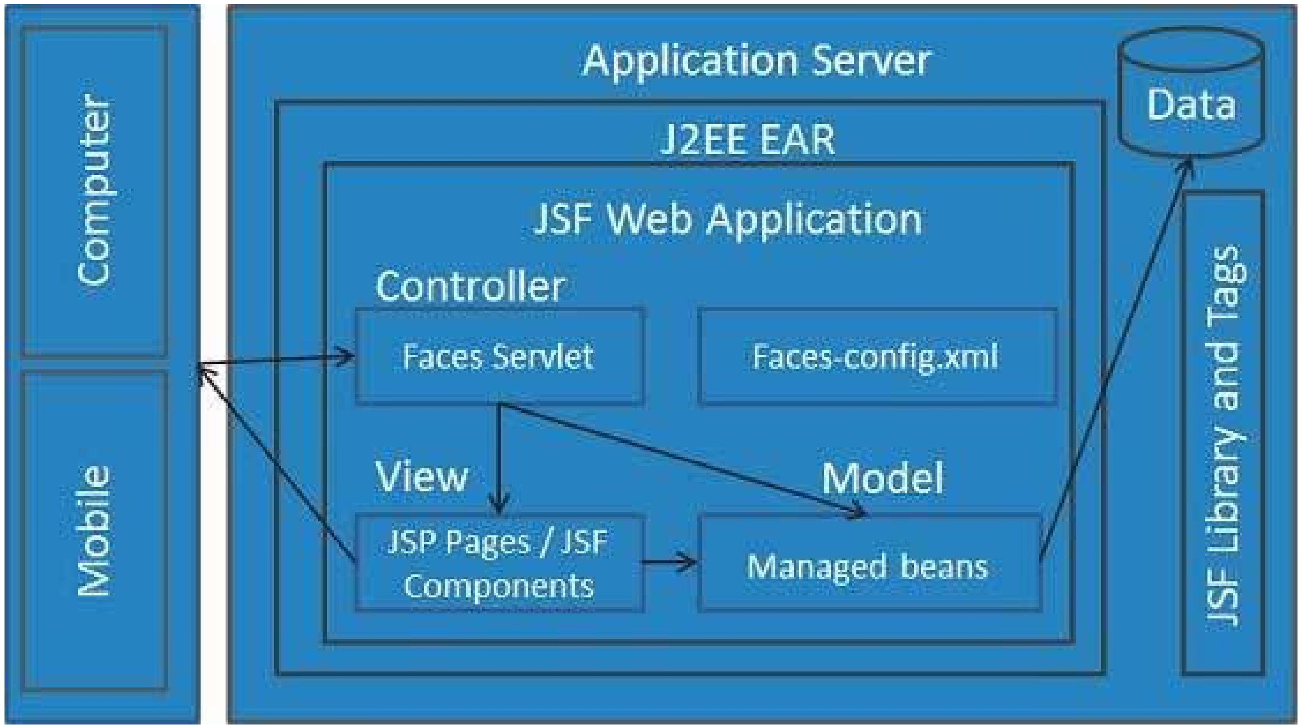
JSF Architecture - Model

- Again, it is an MVC architecture, where the model is a *managed bean*, which in turn may interact with DAO classes or EJB's (e.g. session beans acting as facades for entity classes) which are the real underlying model in our application, but in JSF the managed beans are also considered part of the model



JSF Architecture - View

- The view is made up of the UI components on the page, which are rendered by JSF
- Generally you use either Facelets (special tag library for use in HTML pages) or JSP



JSF Architecture - Controller

- The controller / router which routes incoming requests and selects a view for display. In JSF this controller, the *FacesServlet*, is provided by the framework
- Defined in web.xml

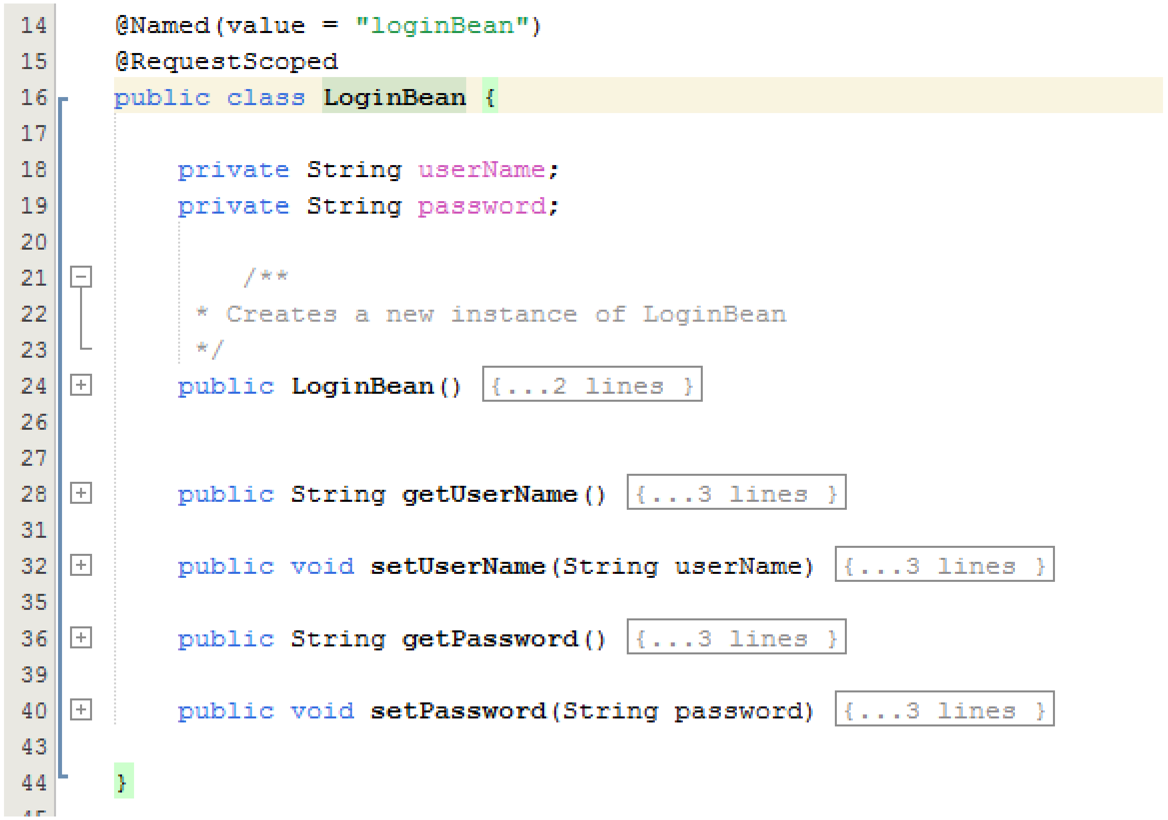
```
<servlet>
     <servlet-name>Faces Servlet</servlet-name>
     <servlet-class>jakarta.faces.webapp.FacesServlet</servlet-class>
     <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
     <servlet-name>Faces Servlet</servlet-name>
     <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
```

FacesServlet

- JSF has a front controller servlet called FacesServlet
- FacesServlet performs the role of brokering the incoming requests from clients to the right places. As mentioned earlier, JSF comes with reusable Web components that can be used to develop user interfaces
- These UI components can be associated with objects called *managed beans*
- *These managed beans handle the logic* for the application and interact with back-end systems or components like EJBs

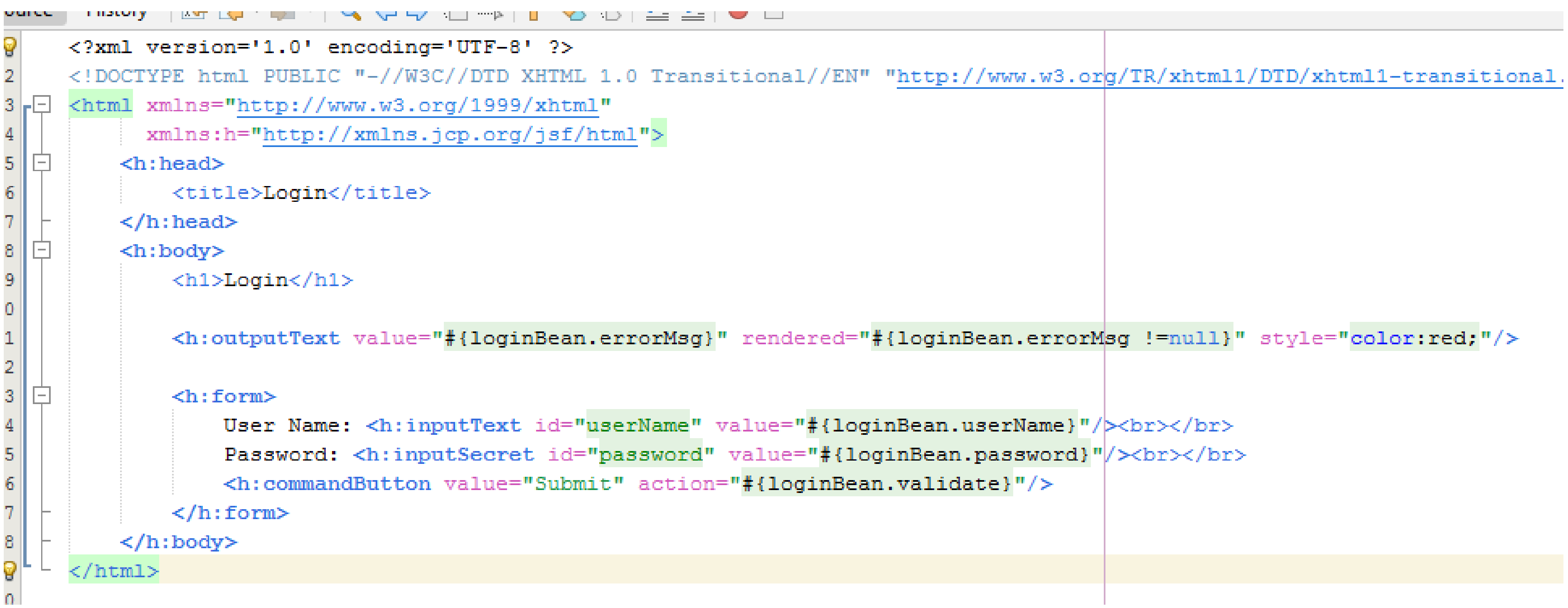
First JSF application

- Create a new Java class, LoginBean.java
- Store in beans folder
- Annotate as shown
- Give it 2 properties:
  - String userName
  - String password
- Create getters and setters



```
@Named(value = "loginBean")
@RequestScoped
public class LoginBean {

    private String userName;
    private String password;

        /**
     * Creates a new instance of LoginBean
     */
    public LoginBean() {...2 lines }


    public String getUserName() {...3 lines }

    public void setUserName(String userName) {...3 lines }

    public String getPassword() {...3 lines }

    public void setPassword(String password) {...3 lines }

}
```

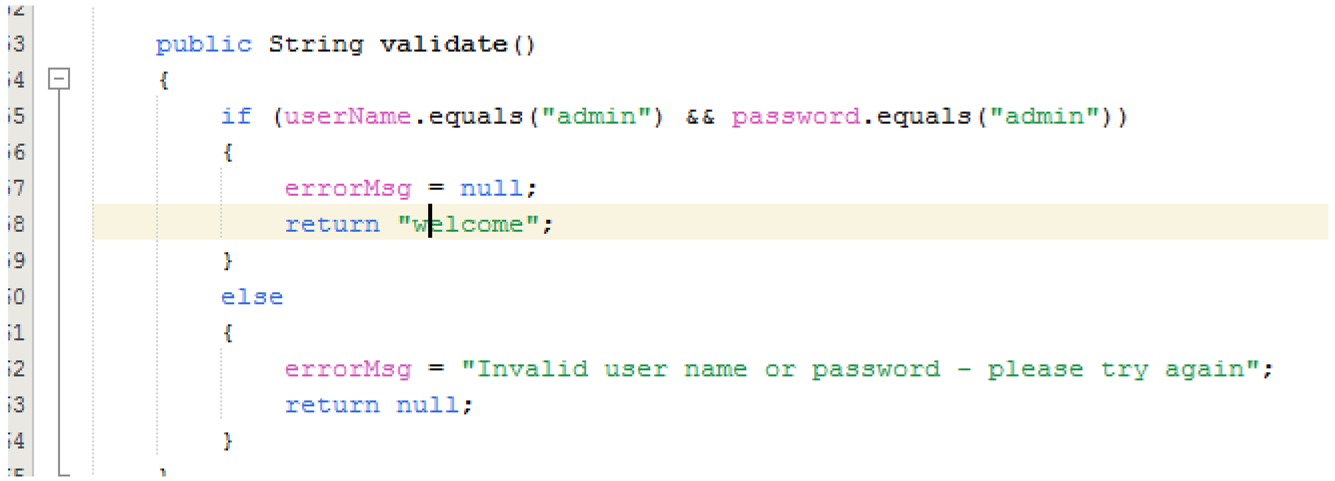
Edit index.xhtml



```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Login</title>
    </h:head>
    <h:body>
        <h1>Login</h1>

        <h:outputText value="#{loginBean.errorMsg}" rendered="#{loginBean.errorMsg !=null}" style="color:red;"/>

        <h:form>
            User Name: <h:inputText id="userName" value="#{loginBean.userName}"/><br></br>
            Password: <h:inputSecret id="password" value="#{loginBean.password}"/><br></br>
            <h:commandButton value="Submit" action="#{loginBean.validate}"/>
        </h:form>
    </h:body>
</html>
```

Update LoginBean.java

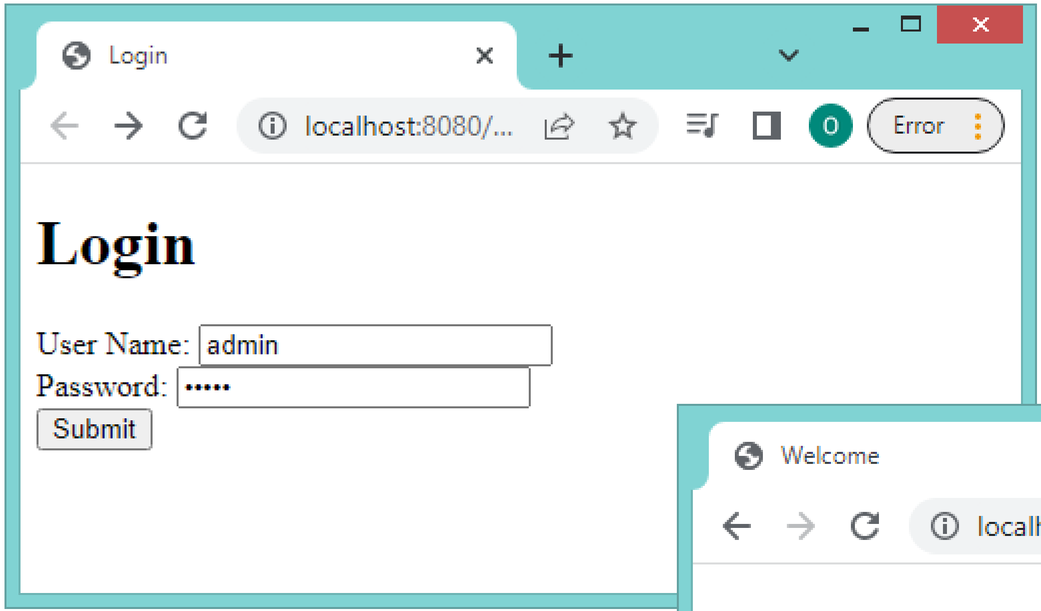
- Need to add the String property errorMsg and it's getter and setter methods
- Also note that we are calling a validate method in the bean when we submit the form



```
public String validate()
{
    if (userName.equals("admin") && password.equals("admin"))
    {
        errorMsg = null;
        return "welcome";
    }
    else
    {
        errorMsg = "Invalid user name or password - please try again";
        return null;
    }
}
```

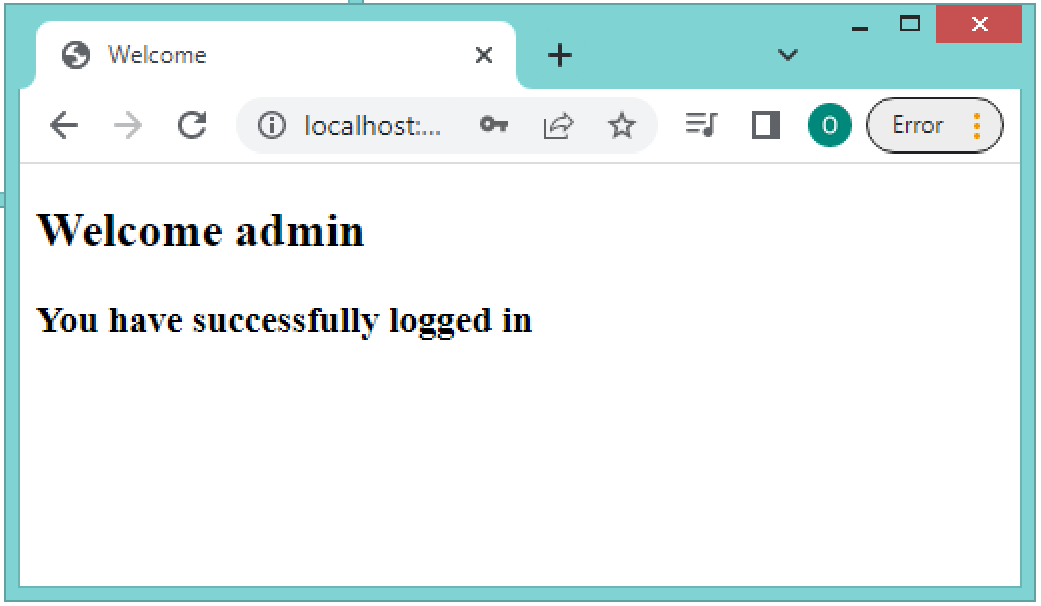
- Note also that this method returns a string, which is used by JSF to navigate to a specific page, in this case "welcome.xhtml"

Run the application



Login

User Name: admin
Password: •••••
Submit



Welcome admin

You have successfully logged in

Managed Bean

- Just a Java (POJO) class inside a container that the developer doesn't have to instantiate manually
  - Sits behind the JSF page

- @Named – gives the bean we can refer to from our xhtml files
  - Properties of bean must have getters and setters
- @RequestScoped
  - An instance is created when requested and destroyed when page has loaded in the browser

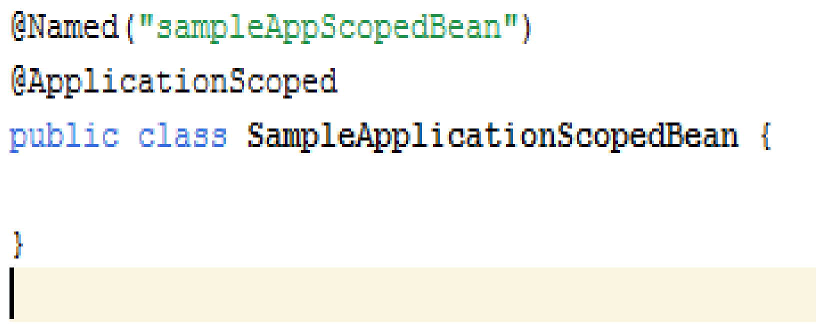
Managed bean scope = lifespan

- Long-lived beans
  - Application-scoped
  - Session-scoped
- Short-lived beans
  - Request-scoped
  - View-scoped
  - Flash-scoped

Application scoped

- Created on load and kept alive for duration of application

```
import javax.enterprise.context.ApplicationScoped;
```



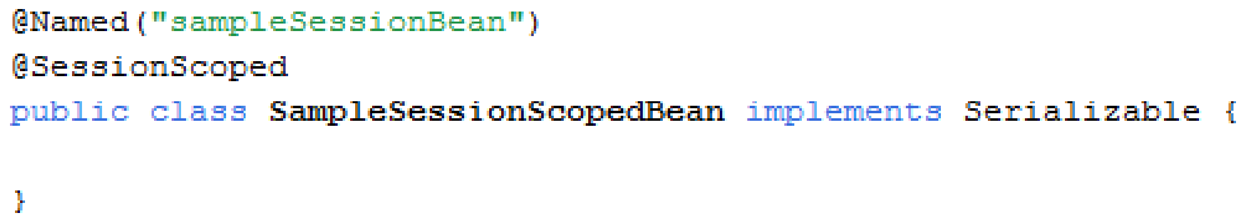
```
@Named("sampleAppScopedBean")
@ApplicationScoped
public class SampleApplicationScopedBean {

}
```

- Only one instance will exist in the application

Session scoped

- Must implement *Serializable*
- One per HTTP (user’s browser) session

```
import javax.enterprise.context.SessionScoped;
```

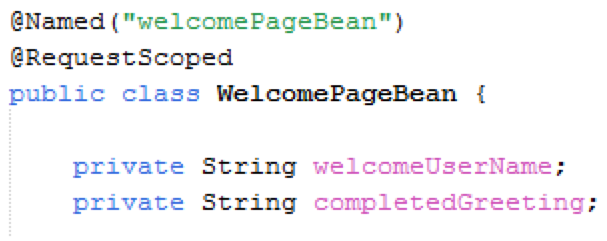


```
@Named("sampleSessionBean")
@SessionScoped
public class SampleSessionScopedBean implements Serializable {

}
```

Request scoped

- Instantiated at beginning of JSF page request and destroyed when page is loaded

```
import javax.enterprise.context.RequestScoped;
```

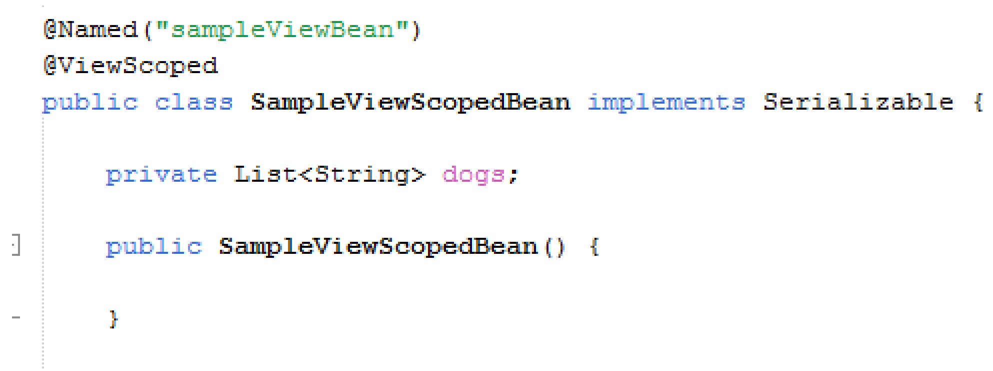


```
@Named("welcomePageBean")
@RequestScoped
public class WelcomePageBean {

    private String welcomeUserName;
    private String completedGreeting;
```

View scoped

- Intermediate between session and request scope
- Lives past a single page load, but destroyed if you navigate to a different page
- Useful for a *conversation* with user / set of interactions on a single page

import javax.faces.view.ViewScoped;



```java
@Named("sampleViewBean")
@ViewScoped
public class SampleViewScopedBean implements Serializable {

    private List<String> dogs;

    public SampleViewScopedBean() {

    }
```

Flash scope

- Mostly used for transmitting data between JSF pages
  - When navigating between pages
  - Example – see
    - index.xhtml
    - WelcomePageBean navigateToFlashPage method
    - flashscope.xhtml
    - Puts a variable on the flash scope to be picked up by the flashscope.xhtml page, using the code:

```
<h:outputText value="#{flash.transmittedVariable}"/>
```

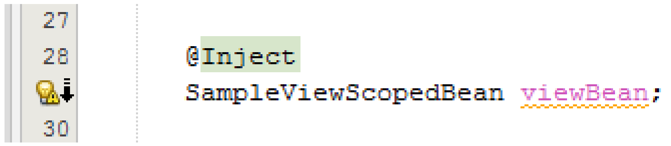
Initialising data in managed bean

- See the *SampleViewScopedBean.java* file
- The initialisation of the data is not done in the *getDogs()* getter method, but in the *initDogs()* method, annotated with *@PostConstruct*
  - Can have only one such method, and it has to return void
- Only want to do it only once when bean is created, for example by calling DAO object or session façade for entity bean
- The matching method for when bean is destroyed, is the *cleanUp* method

- Note also the retrieval and invalidation of the session object – you might use this in a logout or session timeout situation

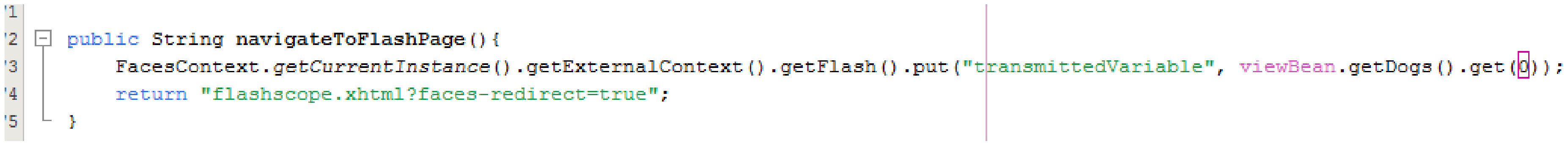
Injecting into beans

- We can inject other beans, including other managed beans, into managed beans, e.g. in WelcomePageBean:



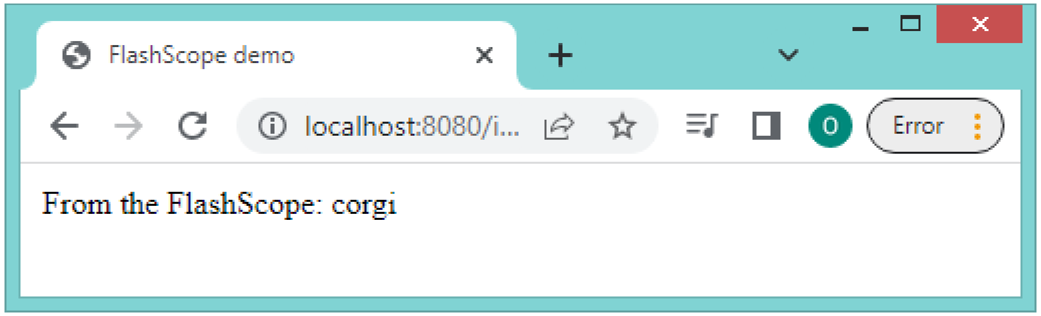
```
@Inject
SampleViewScopedBean viewBean;
```

- And can then use it in the navigateToFlashPage() method



```
public String navigateToFlashPage(){
    FacesContext.getCurrentInstance().getExternalContext().getFlash().put("transmittedVariable", viewBean.getDogs().get(0));
    return "flashscope.xhtml?faces-redirect=true";
}
```

- Clicking on the Navigate button in index.xhtml results in

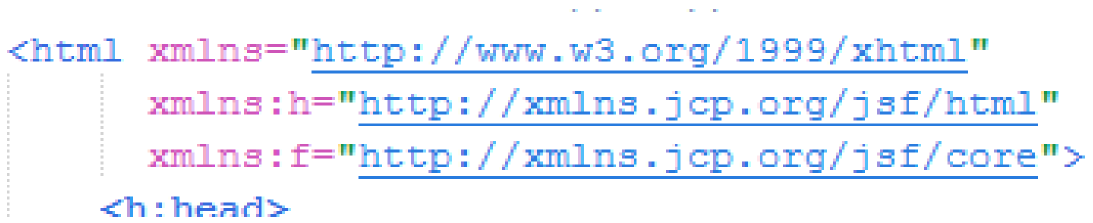


From the FlashScope: corgi

- To inject session façade beans like the ones created in the AffableBean example, we could use the @Inject or the @EJB annotation

facelets

- There is a library of jsf core components in addition to the ones already use, which you include using the ../jsf/core command, e.g.



```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
```

jsf/html includes tags that represent common HTML user interface components

jsf/core tag library defines tags that perform core actions and are independent of a specific (e.g. HTML) rendering kit

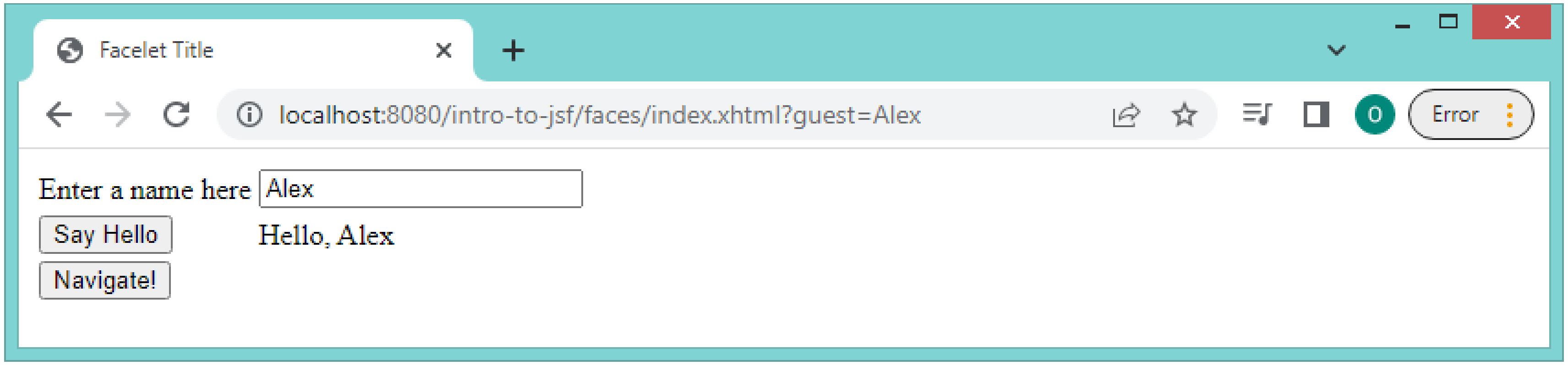
There are other libraries available too, such as PrimeFaces

facelet code tag

- See how the viewParam tag lets us process a request parameter in the current view

```
<f:metadata>
    <f:viewParam name="guest" value="#{welcomePageBean.welcomeUserName}"
    <f:viewAction action="#{welcomePageBean.sayHello}"/>
</f:metadata>
```

- It gets the request parameter and uses it to set the welcomeUserName property on the managed bean
- It then calls theviewAction which ends up setting the value of the <h>outputText



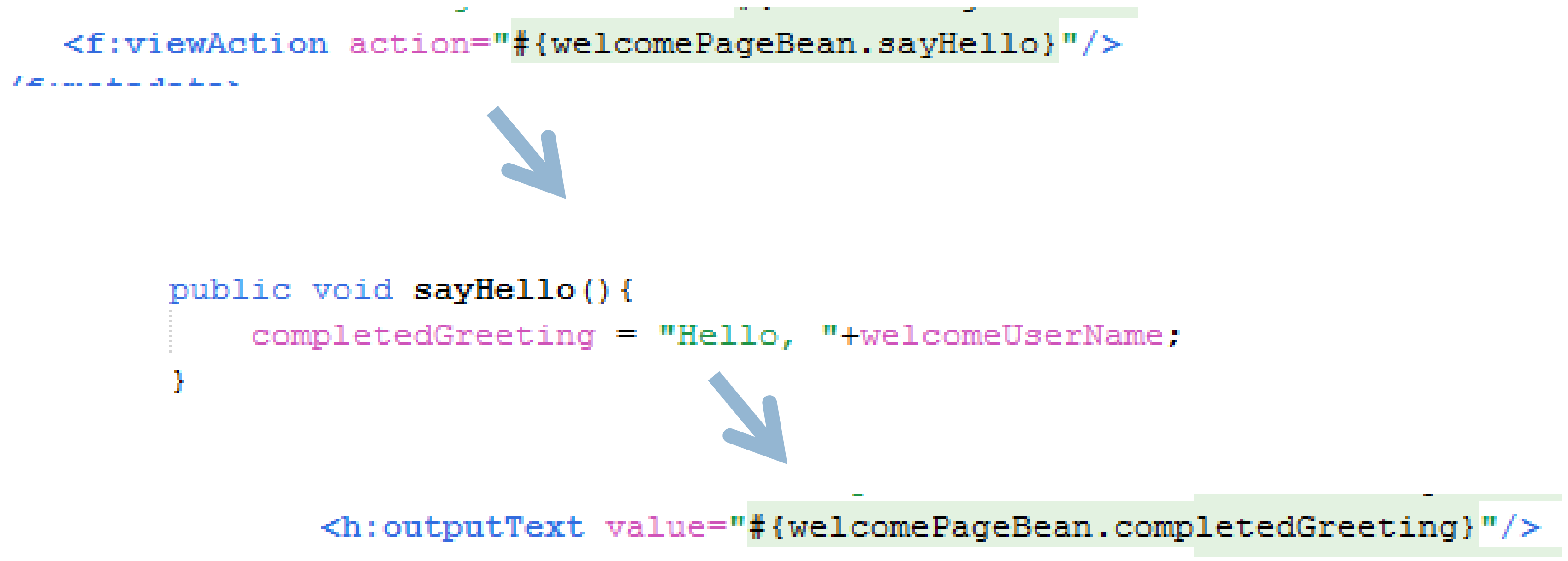
localhost:8080/intro-to-jsf/faces/index.xhtml?guest=Alex

Enter a name here Alex

Say Hello   Hello, Alex

Navigate!

Sequence of action events



```
<f:viewAction action="#{welcomePageBean.sayHello}"/>
```

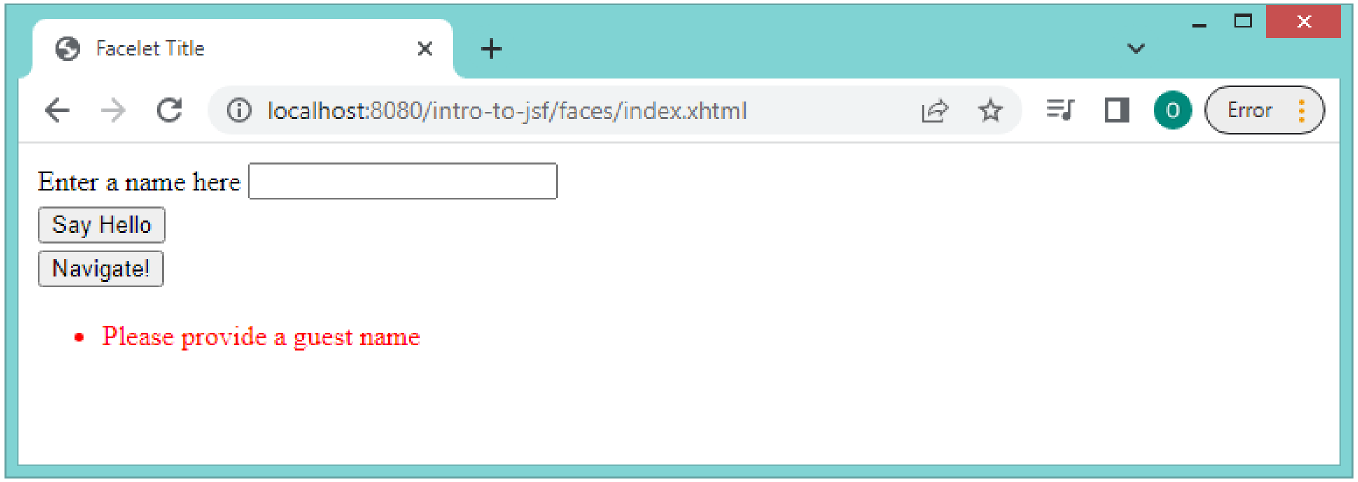
```
public void sayHello(){
    completedGreeting = "Hello, "+welcomeUserName;
}
```

```
<h:outputText value="#{welcomePageBean.completedGreeting}"/>
```

If the URL parameter is missing

- If the parameter is empty and the call is not from a postback from a bean method then display *requiredMessage*
- *When postback is true then page has been refreshed, in which case we don't want the parameter to be required*

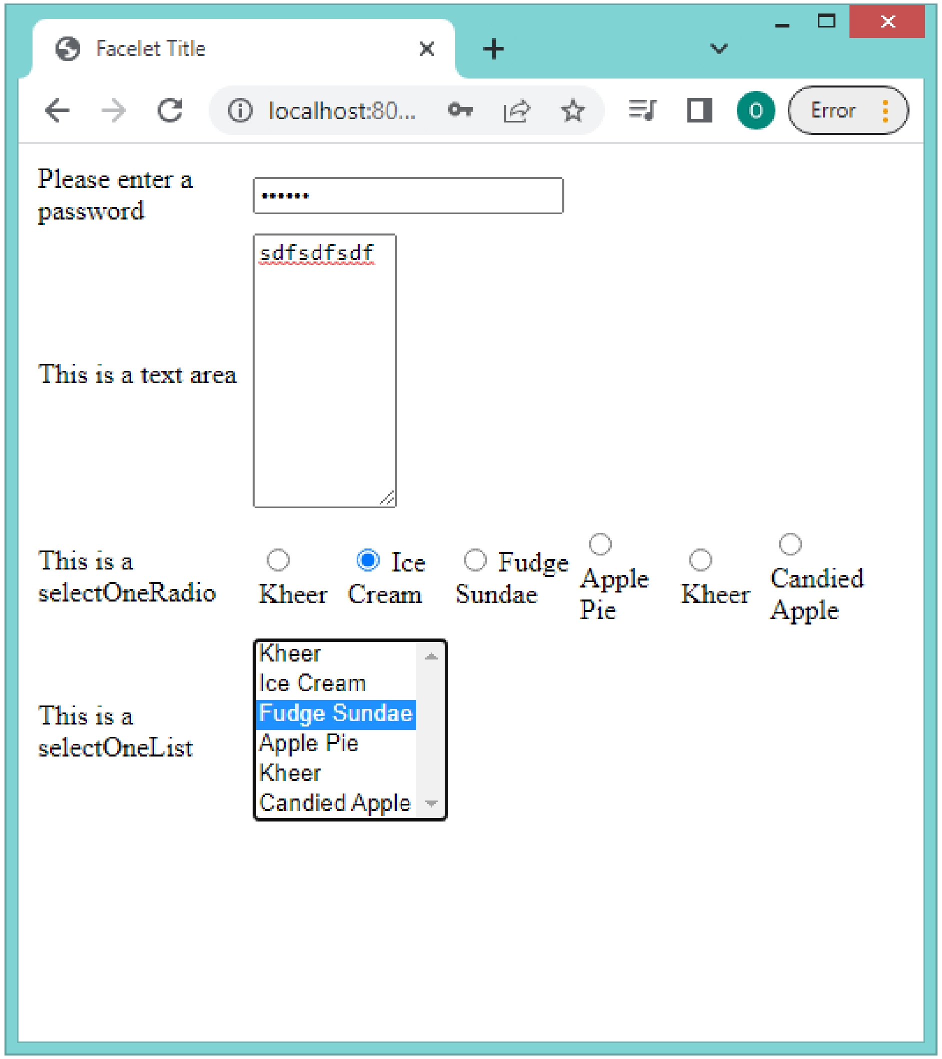
```
required="#{!facesContext.postback}" requiredMessage="Please provide a guest name"/>
```



- Please provide a guest name

inputComponents.xhtml

- This page demonstrates how to pull data from a managed bean into user interface components such as radio buttons, list boxes,..
- Note the use of the value property of the input components to tie the selected item to a property of the backing bean



Please enter a password ••••••

This is a text area sdfsdfsdf

This is a selectOneRadio ○ Kheer [● Ice Cream] ○ Fudge Sundae ○ Apple Pie ○ Kheer ○ Candied Apple

This is a selectOneList

Kheer
Ice Cream
[✓ Fudge Sundae]
Apple Pie
Kheer
Candied Apple